

Part I: Divide and Conquer

Lecture 3: The Polynomial Multiplication Problem

Ahmed Khademzadeh
<http://khademzadeh.mshdiau.ac.ir>
Azad University of Mashhad



The slides are borrowed from Ke (Kevin) Yi. Thanks to him for his benevolence.

Objective and Outline

Objective: Show an second example of divide-and-conquer

Outline:

- The polynomial multiplication problem
- A brute force algorithm
- A first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Divide-and-Conquer

Divide

- Divide a given problem into two or more subproblems (ideally of approximately equal size)

Conquer

- Solve each subproblem (directly if small enough or **recursively**)

Combine

- Combine the solutions of the subproblems into a global solution

Outline:

- The general form of divide-and-conquer
- The polynomial multiplication problem
- A brute force algorithm
- A first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

The Polynomial Multiplication Problem

Definition (Polynomial Multiplication Problem)

Given two polynomials

$$A(x) = a_0 + a_1x + \cdots + a_nx^n$$

$$B(x) = b_0 + b_1x + \cdots + b_mx^m$$

Compute the **product** $A(x)B(x)$

Example

$$A(x) = 1 + 2x + 3x^2$$

$$B(x) = 3 + 2x + 2x^2$$

$$A(x)B(x) = 3 + 8x + 15x^2 + 10x^3 + 6x^4$$

- Assume that the coefficients a_i and b_i are stored in arrays $A[0 \dots n]$ and $B[0 \dots m]$
- **Cost**: number of scalar multiplications and additions

What do we need to compute exactly?

Define

- $A(x) = \sum_{i=0}^n a_i x^i$
- $B(x) = \sum_{i=0}^m b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{n+m} c_k x^k$

Then

$$c_k = \sum_{0 \leq i \leq n, 0 \leq j \leq m, i+j=k} a_i b_j, \quad \text{for all } 0 \leq k \leq m+n$$

Definition

The vector $(c_0, c_1, \dots, c_{m+n})$ is the **convolution** of the vectors (a_0, a_1, \dots, a_n) and (b_0, b_1, \dots, b_m)

We need to calculate convolutions. This is a major problem in digital signal processing

Outline:

- The general form of divide-and-conquer
- The polynomial multiplication problem
- A brute force algorithm
- A first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Direct (Brute Force) Approach

To ease analysis, assume $n = m$.

- $A(x) = \sum_{i=0}^n a_i x^i$
- $B(x) = \sum_{i=0}^n b_i x^i$
- $C(x) = A(x)B(x) = \sum_{k=0}^{2n} c_k x^k$ with

$$c_k = \sum_{0 \leq i, j \leq n, i+j=k} a_i b_j, \quad \text{for all } 0 \leq k \leq 2n$$

Direct approach: Compute all c_k 's using the formula above

- Total number of multiplications: $\Theta(n^2)$
- Total number of additions: $\Theta(n^2)$
- Complexity: $\Theta(n^2)$

Outline:

- The general form of divide-and-conquer
- The polynomial multiplication problem
- A brute force algorithm
- A first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Divide-and-Conquer: Divide

Assume n is a power of 2

Define

$$A_0(x) = a_0 + a_1x + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1}$$

$$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + \cdots + a_nx^{\frac{n}{2}}$$

$$A(x) = A_0(x) + A_1(x)x^{\frac{n}{2}}$$

Similarly, define $B_0(x)$ and $B_1(x)$ such that

$$B(x) = B_0(x) + B_1(x)x^{\frac{n}{2}}$$

$$\begin{aligned} A(x)B(x) &= A_0(x)B_0(x) + A_0(x)B_1(x)x^{\frac{n}{2}} \\ &\quad + A_1(x)B_0(x)x^{\frac{n}{2}} + A_1(x)B_1(x)x^n \end{aligned}$$

The original problem (of size n) is divided into 4 problems of input size $n/2$

Example

$$A(x) = 2 + 5x + 3x^2 + x^3 - x^4$$

$$B(x) = 1 + 2x + 2x^2 + 3x^3 + 6x^4$$

$$A(x)B(x) = 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 \\ + 19x^6 + 3x^7 - 6x^8$$

$$A_0(x) = 2 + 5x, A_1(x) = 3 + x - x^2, A(x) = A_0(x) + A_1(x)x^2 \\ B_0(x) = 1 + 2x, B_1(x) = 2 + 3x + 6x^2, B(x) = B_0(x) + B_1(x)x^2$$

$$A_0(x)B_0(x) = 2 + 9x + 10x^2$$

$$A_1(x)B_1(x) = 6 + 11x + 19x^2 + 3x^3 - 6x^4$$

$$A_0(x)B_1(x) = 4 + 16x + 27x^2 + 30x^3$$

$$A_1(x)B_0(x) = 3 + 7x + x^2 - 2x^3$$

$$A_0(x)B_1(x) + A_1(x)B_0(x) = 7 + 23x + 28x^2 + 28x^3$$

$$A_0(x)B_0(x) + (A_0(x)B_1(x) + A_1(x)B_0(x))x^2 + A_1(x)B_1(x)x^4 \\ = 2 + 9x + 17x^2 + 23x^3 + 34x^4 + 39x^5 + 19x^6 + 3x^7 - 6x^8$$

Divide-and-Conquer: Conquer

Conquer: Solve the four subproblems

- compute

$$A_0(x)B_0(x), \quad A_0(x)B_1(x), \quad A_1(x)B_0(x), \quad A_1(x)B_1(x)$$

by recursively calling the algorithm **4 times**

Combine

- add the following four polynomials

$$\begin{aligned} &A_0(x)B_0(x) \quad + A_0(x)B_1(x)x^{\frac{n}{2}} \\ &\quad \quad \quad + A_1(x)B_0(x)x^{\frac{n}{2}} \\ &\quad \quad \quad + A_1(x)B_1(x)x^n \end{aligned}$$

- takes $O(n)$ operations (Why?)

The First Divide-and-Conquer Algorithm

PolyMulti1(A(x), B(x))

begin

$$A_0(x) = a_0 + a_1x + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$$

$$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + \cdots + a_nx^{\frac{n}{2}};$$

$$B_0(x) = b_0 + b_1x + \cdots + b_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$$

$$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1}x + \cdots + b_nx^{\frac{n}{2}};$$

$$U(x) = \text{PolyMulti1}(A_0(x), B_0(x));$$

$$V(x) = \text{PolyMulti1}(A_0(x), B_1(x));$$

$$W(x) = \text{PolyMulti1}(A_1(x), B_0(x));$$

$$Z(x) = \text{PolyMulti1}(A_1(x), B_1(x));$$

$$\text{return } (U(x) + [V(x) + W(x)]x^{\frac{n}{2}} + Z(x)x^n)$$

end

Analysis of Running Time

Assume that n is a power of 2, $n = 2^h$

$$T(n) \leq 4T\left(\frac{n}{2}\right) + cn; T(1) = 1$$

$$\begin{aligned}T(n) &\leq 4 \left[4T\left(\frac{n}{2^2}\right) + c\frac{n}{2} \right] + cn = 4^2 T\left(\frac{n}{2^2}\right) + (1+2)cn \\ &\leq 4^2 \left[4T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2} \right] + (1+2)cn = 4^3 T\left(\frac{n}{2^3}\right) + (1+2+2^2)cn \\ &\leq \dots = 4^i T\left(\frac{n}{2^i}\right) + \sum_{j=0}^{i-1} 2^j cn \quad (\text{induction}) \\ &\leq \dots = 4^h T\left(\frac{n}{2^h}\right) + \sum_{j=0}^{h-1} 2^j cn \\ &\leq n^2 T(1) + cn(n-1) \quad (\text{since } n = 2^h \text{ and } \sum_{j=0}^{h-1} 2^j = 2^h - 1 = n - 1) \\ &= O(n^2)\end{aligned}$$

Same order as the brute force approach! No improvement!

Solving the Recurrence in General

Assume $n = b^h$

$T(n) \leq aT\left(\frac{n}{b}\right) + cn; T(1) = 1; a \geq 1, b > 1$ are constants

$$\begin{aligned}T(n) &\leq a \left[a T\left(\frac{n}{b^2}\right) + c \frac{n}{b} \right] + c n \\&\leq a^2 T\left(\frac{n}{b^2}\right) + \left(1 + \frac{a}{b}\right) c n \\&\leq a^2 \left[a T\left(\frac{n}{b^3}\right) + c \frac{n}{b^2} \right] + \left(1 + \frac{a}{b}\right) c n \\&\leq a^3 T\left(\frac{n}{b^3}\right) + \left(1 + \frac{a}{b} + \left[\frac{a}{b}\right]^2\right) c n \\&\vdots \\&\leq a^h T\left(\frac{n}{b^h}\right) + \sum_{j=0}^{h-1} \left[\frac{a}{b}\right]^j c n\end{aligned}$$

We have

$$a^h = (b^{\log_b a})^h = b^{h \log_b a} = (b^h)^{\log_b a} = n^{\log_b a},$$

and if $a > b$,

$$\sum_{j=0}^{h-1} \left[\frac{a}{b} \right]^j = \frac{(a/b)^h - 1}{a/b - 1} \leq \frac{(a/b)^h}{a/b - 1} = \frac{n^{\log_b a}/n}{a/b - 1}$$

Recall that

$$T(n) \leq a^h T\left(\frac{n}{b^h}\right) + \sum_{j=0}^{h-1} \left[\frac{a}{b} \right]^j c n$$

Hence

$$T(n) = O(n^{\log_b a} T(1) + n^{\log_b a}/n \cdot n) = O(n^{\log_b a})$$

Example

For our first divide-and-conquer algorithm, $a = 4$, $b = 2$, $\log_2 4 = 2$, so $T(n) = O(n^2)$.

If $a = b$,

$$a^h = n^{\log_b a} = n,$$

and

$$\sum_{j=0}^{h-1} \left[\frac{a}{b}\right]^j = \sum_{j=0}^{h-1} 1 = h$$

So

$$T(n) \leq a^h T\left(\frac{n}{b^h}\right) + \sum_{j=0}^{h-1} \left[\frac{a}{b}\right]^j c n = O(n + hn) = O(n \log n)$$

Example

In the divide-and-conquer algorithm for **maximum contiguous subarray** problem, $a = 2$, $b = 2$, so $T(n) = O(n \log n)$.

If $a < b$,

$$a^h = n^{\log_b a} = o(n),$$

and

$$\sum_{j=0}^{h-1} \left[\frac{a}{b}\right]^j = \frac{1 - (a/b)^h}{1 - a/b} = O(1)$$

So

$$T(n) \leq a^h T\left(\frac{n}{b^h}\right) + \sum_{j=0}^{h-1} \left[\frac{a}{b}\right]^j c n = O(n^{\log_b a} + n) = O(n)$$

Example

Linear-time selection (will see soon)

Outline:

- The general form of divide-and-conquer
- The polynomial multiplication problem
- A brute force algorithm
- A first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

Two Observations

Observation 1:

What we really need are the following 3 terms:

$$A_0B_0, A_0B_1 + A_1B_0, A_1B_1?$$

Instead of the following 4 terms:

$$A_0B_0, A_0B_1, A_1B_0, A_1B_1?$$

Observation 2:

The three terms can be obtained using only 3 multiplications:

$$Y = (A_0 + A_1)(B_0 + B_1)$$

$$U = A_0B_0$$

$$Z = A_1B_1$$

- U and Z are what we originally wanted
- $A_0B_1 + A_1B_0 = Y - U - Z$

The Second Divide-and-Conquer Algorithm

PolyMulti2(A(x), B(x))

begin

$$A_0(x) = a_0 + a_1x + \cdots + a_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$$

$$A_1(x) = a_{\frac{n}{2}} + a_{\frac{n}{2}+1}x + \cdots + a_nx^{n-\frac{n}{2}};$$

$$B_0(x) = b_0 + b_1x + \cdots + b_{\frac{n}{2}-1}x^{\frac{n}{2}-1};$$

$$B_1(x) = b_{\frac{n}{2}} + b_{\frac{n}{2}+1}x + \cdots + b_nx^{n-\frac{n}{2}};$$

$$Y(x) = \text{PolyMulti2}(A_0(x) + A_1(x), B_0(x) + B_1(x));$$

$$U(x) = \text{PolyMulti2}(A_0(x), B_0(x));$$

$$Z(x) = \text{PolyMulti2}(A_1(x), B_1(x));$$

$$\text{return } (U(x) + [Y(x) - U(x) - Z(x)]x^{\frac{n}{2}} + Z(x)x^{2\frac{n}{2}})$$

end

Running Time of the Modified Algorithm

$$T(n) \leq 3T\left(\frac{n}{2}\right) + cn; T(1) = 1$$

$a = 3, b = 2$, hence,

$$T(n) = O(n^{\log_2 3}) = O(n^{1.585})$$

- The second method is much better!

Outline:

- The general form of divide-and-conquer
- The polynomial multiplication problem
- A brute force algorithm
- A first divide-and-conquer algorithm
- An improved divide-and-conquer algorithm
- Remarks

- The divide-and-conquer approach does not always give you the best solution
 - Our original algorithm was just as bad as brute force
- There is actually an $O(n \log n)$ solution to the polynomial multiplication problem
 - It involves using the **Fast Fourier Transform** algorithm as a subroutine
 - The FFT is another classic divide-and-conquer algorithm (Chapt 30 in CLRS gives details) (**not required**)
- The idea of using 3 multiplications instead of 4 is used in large-integer multiplications
 - A similar idea is the basis of the classic **Strassen matrix multiplication algorithm** (CLRS, Chapter 28.2)